Proceeding of the 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space: i-SAIRAS 2001, Canadian Space Agency, St-Hubert, Quebec, Canada, June 18-22, 2001.

VIPER: Virtual Intelligent Planetary Exploration Rover

Laurence Edwards Lorenzo Flückiger* **Laurent Nguyen†** **Richard Washington‡**
Autonomy and Robotics Area, NASA Ames Research Center, Moffett Field, CA 94035
{ edwards | lorenzo | nguyen | richw } @artemis.arc.nasa.gov

Keywords: Simulation, 3D visualization, plan execution, planetary rovers.

Abstract

Simulation and visualization of rover behavior are critical capabilities for scientists and rover operators to construct, test, and validate plans for commanding a remote rover. The VIPER system links these capabilities, using a high-fidelity virtual-reality (VR) environment, a kinematically accurate simulator, and a flexible plan executive to allow users to simulate and visualize possible execution outcomes of a plan under development.

This work is part of a larger vision of a science-centered rover control environment, where a scientist may inspect and explore the environment via VR tools, specify science goals, and visualize the expected and actual behavior of the remote rover.

The VIPER system is constructed from three generic systems, linked together via a minimal amount of customization into the integrated system. The complete system points out the power of combining plan execution, simulation, and visualization for envisioning rover behavior; it also demonstrates the utility of developing generic technologies, which can be combined in novel and useful ways.

1 Introduction

Imagine trying to drive when your vehicle only does approximately what you command, you only catch occasional glimpse of your environment, 20 minutes pass between your command and the vehicle's response, and to top it off, you don't really know how your vehicle works. This is the world of scientist-directed planetary rover exploration.

Planetary rovers are scientific tools for exploring an unknown world. One focus of the Autonomy and Robotics Area (ARA) at the NASA Ames Research Center is to design and develop the tools and techniques that allow scientists to control a rover efficiently and effectively. This presents challenges both in the user interface and in the underlying rover control methods.

One important element of the planetary rover control is the ability to simulate and visualize possible execution outcomes of a plan under development. We have developed the VIPER system, which links plan execution, rover simulation, and a high-fidelity, realistic virtual-reality (VR) environment. This system is one part of a larger architectural design under development that includes tools for science goal specification and plan generation.

The ultimate vision for the overall architecture is that scientists at "mission control," and potentially elsewhere in the world, will both specify and observe the rover's operation as well as science products through the VR environment. The scientists can examine physical features of the environment (distance, volume, cross-sections) and specify science-level goals, for example to go to a rock and drill a small sample. These goals are then interactively refined at mission control with the help of a planning and scheduling system, adding constraints of rover motion, resources, and time to arrive at a final plan. Once the plan is ready, it is communicated to the rover.

On board the rover, the plan is executed by testing and monitoring conditions on time, resources, and rover and environmental state. The same plan executed multiple times may produce many different behaviors, based on the initial conditions and the variability of the rover's interactions with the environment.

The VIPER system allows users to simulate and visualize possible execution outcomes of a plan under development. We have developed a kinematically accurate simulator of the rover that allows the sci-

*NASA contractor with QSS.

†Author's current address is LightLogic, Inc., 8674 Thornton Avenue, Newark, CA 94560.

‡NASA contractor with RIACS.

entists and rover engineers to understand more concretely the effects of plan actions. The plan execution system commands the simulator, which in turn controls a model of the rover within the VR environment. This provides an immediate connection between the plan and the resulting execution behavior. Each of the technologies was developed as a generic system in a separate context from this system; the resulting system was then assembled quickly with relatively minor specializations.

This paper offers two contributions. It highlights the power of combining plan execution, simulation, and visualization for the problem of rover control. It also demonstrates the advantages of using generic technologies for rapid development of useful systems. In the remainder of the paper, we will describe the overall system history and organization, followed by a more detailed description of each component. We will then present an example to illustrate the operation of the system. Finally, we will discuss the ongoing work to extend the system.

2 System Background and Overview

The VIPER system is a science-oriented rover control system that comprises three technologies: plan execution, simulation, and visualization. In this section we describe the background of the individual technologies, then describe the organization of the VIPER system.

Virtual Reality Interfaces Our design for a rover-control architecture is built around the premise that a scientist should be able to specify science goals and to control rover activities with little or no assistance from rover engineers and technologists. This goal of a science-oriented system stems from a line of robotic tele-operation research begun in the early '90s at NASA Ames exploring the use of Virtual Reality (VR) user interfaces for the control of robotic mechanisms. The visualization technology also derives from this line of work.

The development of VR user interfaces for robotic control was motivated by the difficulty of operating complex high degree of freedom mechanisms in unknown, remote, hazardous environments. In the absence of a robust fully autonomous capability, the ability of human operators to understand quickly and accurately a mechanism's relationship to its environment becomes pivotal. It was hypothesized that high fidelity, interactive, 3D (i.e., VR) representations of a robot and its environment would effectively leverage the human visual system to maximize the bandwidth of human-machine communication, and provide operators with immediate visceral situational understanding.

The first operational system resulting from this research thrust was the Virtual Environment Vehicle Interface (VEVI) [Piguet *et al.*, 1995]. VEVI was designed to be a flexible re-configurable VR user interface for robotic mechanisms. Application specific interfaces could be developed by implementing a communications layer utilizing the provided I/O modules and specifying the geometry and kinematics of a particular mechanism in a configuration file. VEVI was used in a number of projects and tests of mobile robots. The robots controlled with VEVI include the 8-legged "Dante II" walking robot [Bares and Wettergreen, 1999] and the 6-wheeled "Marsokhod" robot [Christian *et al.*, 1997]. Figure 1 shows the Marsokhod rover being controlled in the VEVI interface. VEVI was used in two modes: 1) robot telemetry was used to directly drive the state of the VR environment, and 2) robot operations were first planned offline by simulation in VEVI and then executed. The second mode was found to be more effective due to sensor noise and inaccuracy.

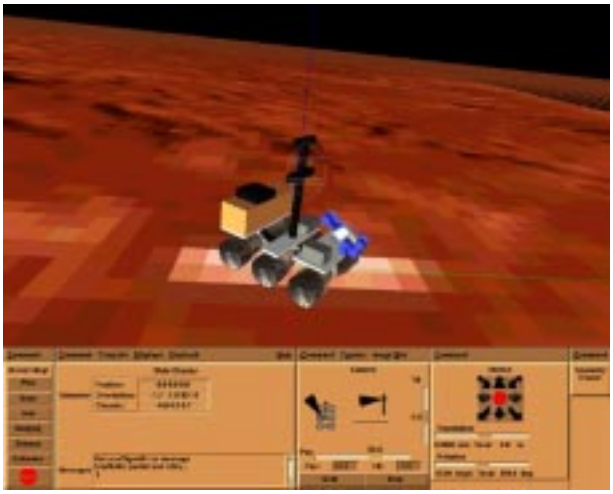

Figure 1: VEVI interface controlling the Marsokhod rover.

In addition to the difficulties cited above, control of robots operating on other planets adds significant communication delays. In this situation the ability to plan the offline in the highest fidelity becomes critical. Scientists involved in the mission are typically not highly trained operators, and tools must be provided that allow them to plan science activities and analyze data in an intuitive and simple manner. To achieve this, NASA Ames developed a VR user interface called "MarsMap" [Stoker *et al.*, 1998] and a high-fidelity 3D-from-stereo terrain reconstruction capability, the "stereo-pipeline." MarsMap was developed for the Mars Pathfinder mission and provided planetary scientists with a suite of simple intuitive tools to plan image acquisition sequences and interrogate stereo-pipeline generated terrain models. The MarsMap environment was highly interactive and immersive, providing enhanced situational understanding.

The Viz system used in this work is a succes-

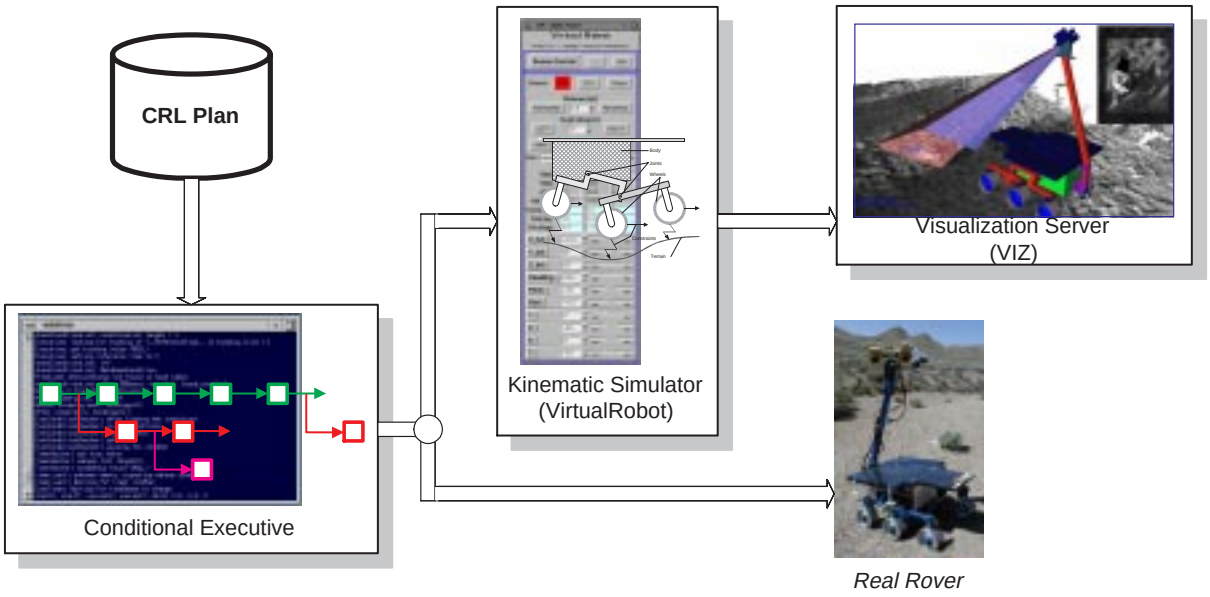

Figure 2: Overview of VIPER plan execution, simulation and visualization.

sor to MarsMap, initially deployed during the Mars Polar Lander mission [Nguyen *et al.*, 2001]. Viz implements an architecture that allows a flexibility and customizability similar in spirit to VEVI and presents the user with a highly interactive immersive environment as in MarsMap.

Robot Behavior Simulation The behavior of the mechanism is reproduced by the VirtualRobot simulator. VirtualRobot was initially developed at the Swiss Federal Institute of Technology, Lausanne (EPFL) by the Virtual Reality and Active Interfaces Group (VRAI) as an interactive tool to control and study any kind of robot manipulator [Flückiger *et al.*, 1998; Flückiger, 1998]. VirtualRobot was based on a generic kinematic generator. The collaboration of the VRAI Group with the Autonomy and Robotics Area of NASA Ames led to extensions of VirtualRobot that enable the simulation of rovers in addition to robot manipulators.

Plan Execution The plan execution component of this work was inspired in part by work on the Remote Agent (RA), an integrated agent architecture developed for spacecraft control and deployed as an experiment on the Deep Space One mission [Bernard *et al.*, 1998; Muscettola *et al.*, 1998]. The rover executive demonstrates advances in conditional execution compared to the RA executive: the language of the RA executive does not accept conditional sequences, which are critical to the success and effectiveness of a rover mission, given the highly variable interactions of the rover and the environment. The current implementation of the rover executive does not, however, attempt to reproduce all of the capabilities of the RA; in particular, multiple concurrent activities, model-based state reconfiguration, and run-time resource selection for state variables are not currently included in our executive.

VIPER system organization The VIPER system comprises the plan execution, simulation, and visualization subsystems (see Figure 2). These components allow the scientists to explore different possible plans and the expected behavior of the robot in the virtual environment.

The plan execution component interprets the command plan, checking conditions and monitoring runtime requirements of the plan. It sends commands to the rover simulation component and receives state information back. The rover simulation component, in turn, simulates the kinematics of the rover and its interactions with the terrain. The simulator sends pose information to the visualization component, which continually updates its environment model and renders the scene for the viewer.

3 Underlying Technologies

The VIPER system is built on generic technologies for each of its subsystems, which are specialized with data or configuration information to work with the particular robotic platform and environment. This allows the system to be used for visualization, test, and design of different, novel, and even imaginary robotic platforms. In particular, parts of this technology have been used to model and simulate the Pathfinder environment, the Mars Polar Lander robotic arm and camera, the NASA Ames

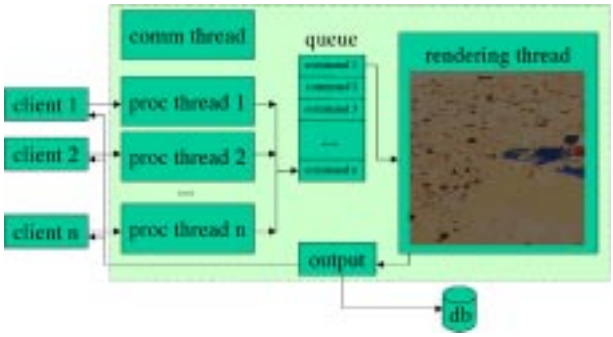

Figure 3: Overview of Viz architecture.

Marsokhod rover, the JPL FIDO rover, and the proposed 2003 MER rover. The entire system has been demonstrated for the NASA Ames K9 rover.

3.1 Viz

The Viz system developed by the ARA is a modular system for distributed 3D visualization. It makes use of high-resolution 3D texture mapped terrain models and articulated 3D models of robot mechanisms to generate a high-fidelity, interactive VR environment. The architecture of Viz facilitates application specific customization and extension of the 3D visualization capabilities, tools and user interaction.

Architecture

The architecture of Viz is based on the client-server paradigm. The core 3D rendering module is implemented as a server to which clients can connect in order to interact with objects in the Virtual Environment (see Figure 3). The server is multi-threaded. A thread is dedicated to each connected client and to the 3D rendering task. The clients are application specific and provide interfaces between the visualization server and the rest of the system (including users). For instance, in the case of robot control, a client can be connected to a robot's sensor and can translate the telemetry into a data format suited for visualization.

A set of predefined messages allows the clients to interact with the server. Examples of messages include add/remove objects, change position/orientation, change color, etc. An automatically generated Python/C++ and Java Viz message interface layer facilitates the development of clients.

The Viz science toolset

A Viz client implementing a set of tools and capabilities was developed by the ARA to support planetary robotic exploration applications. These include a set of science analysis tools for interacting with three dimensional terrain models, and simple simulation capabilities for planning robotic operations.

Viewpoints Viz supports the simultaneous display of multiple viewpoints in separate windows. These viewpoints can be tied to arbitrary reference frames, allowing, for example, the display of the viewpoint from a particular image sensor. This capability is used in VIPER to show images acquired during plan execution.

Measurement tools The measurement tools allow the user to measure the 3D scene. Using a computer mouse controlling a "3D cursor," the user interactively specifies portions of a scene for measurement. These tools are used by scientists to interrogate 3D terrain models reconstructed from sensor data.

Markers This tool allows scientists to annotate terrain by interactively placing objects (typically synthetic) with accompanying text in the 3D scene. Familiar objects of known size are also provided to provide a sense of scale.

Forward kinematic simulation Viz can read Virtual Reality Modeling Language (VRML) files defining the geometry and kinematics of robotic mechanisms. When properly tagged in the file, the Viz system can identify articulated portions of the model and allow articulated movement of mechanisms. This movement can either be driven by user interaction or a Viz client. This capability is used by VIPER to control the motion of a rover model in Viz.

Simulated sensor field-of-view Image sensor fields-of-view can be displayed in Viz as colored semi-transparent sensor-centered pyramids. In addition, a sub-window can be opened displaying a simulated image sensor view of the scene. This allows the user to visualize the features that will be visible from a particular position. With an appropriately tagged VRML model file, image sensors on articulated mounts can be interactively aimed by the user at interesting features to determine pointing information for command generation. In addition, Viz clients can control such articulated sensor mounts and their associated fields-of-views. In VIPER, this is used along with the image sensor viewpoint to provide visual feedback of image acquisition commands.

Customization of Viz for VIPER

The system architecture of Viz allows platform and programming language independent integration of additional capabilities. This flexibility was used to implement the science analysis and planning capability for planetary exploration described above. The only additional work necessary for Viz to be included in the overall VIPER framework was the construction of a rover VRML file including Viz-specific tags indicating articulations. For convenience and ease of use, additional toolbar entries were added to the

Figure 4: Simulation of K9 rover driving over a rock.

existing science toolset interface providing manual control of the rover and its image sensors — this is used for sequence construction. In addition, a button was added to launch the VirtualRobot simulator. The total effort required was a few days.

3.2 VirtualRobot Simulation

VirtualRobot was initially developed as an interactive tool to control and study any kind of robot manipulator [Flückiger *et al.*, 1998; Flückiger, 1998]. The design of VirtualRobot was driven by the following requirements:

- no code writing nor kinematic model is necessary to describe the kinematic behavior of the robot,
- the program is able to handle the kinematics of any robot manipulator structure in real-time[1],
- an intuitive user interface allows both novices and experts to easily manipulate robots in a virtual environment.

For these reasons VirtualRobot was based on a generic kinematic generator: after reading a text file describing the geometry of the robot, the program builds a numerical solver which computes the direct and inverse kinematics of the robot. The robot structure can be serial (as most industrial robots), parallel (as a Stewart platform) or hybrid (mix of the previous two). A robot model is created in a virtual environment and the user is able to interact with the robot using intuitive 3D input devices (like a Space-Mouse) or 6 degree-of-freedom force-feedback devices.

VirtualRobot has been extended to enable the simulation of rovers in addition to robot manipulators. The same kinematic solver can now also accommodate multi-wheeled rovers with passive suspensions driving on uneven terrains (see Figure 4). In addition, rather than being controlled by user input, the kinematic solver responds to inputs coming from any other program through the network.

[1]We consider real-time from a human point of view: refresh rate in the range of 20Hz to 200Hz.

```
rover K9 {

  base {        // the original location of the rover
    pos { -0.6 4.0 0.1 }
    ori { 0.0 0.0 90.0 }
  }
...
  udpdriver "33005"     // how the rover is controlled

  frame 11 {    // position the first boogie
    pos { -0.140 -0.316 0.205 }
    ori { 90.0 0.0 0.0 }
    pred 99
  }

  link 1 {      // main right bogie
    parameters { theta 0.0  r 0.0  alpha 0.0  d 0.0 }
    type 2      // revolute joint
    pred 11     // hierarchy ('1' is child of '11')
    range { -45.0 45.0 }    // limits of the joint
                // graphic object to represent this link
    filename ( "right_main_bogie.wrl")
  }
...
  wheel 1 {     // rear right wheel
    type 1      // type of wheel (geometry)
    pos { 0.0 0.0 0.0 }
    ori { 90.0 0.0 0.0 }
    pred 13
    filename ( "right_wheel.wrl" )
  }
...
  constraint {      // the end-effector '5'
    frame 5         // must follow the input
    sensor ( 0 1 )  // described as sensor '1'
  }
...
} // end of rover description
```

Figure 5: Excerpts of a robot file description used by VirtualRobot to build and simulate a rover

The following two sections briefly review the robot description file and the kinematic solver used in VirtualRobot.

Robot description file

Any robot manipulator, rover or combination of both can be described in a human readable text file which is parsed by VirtualRobot. This file contains all the geometric properties of the mechanical structure to be simulated. The robot structure can be expressed with the Khalil-Kleinfinger formalism [Khalil and Kleinfinger, 1986] (which is also usable for multi-branch structures, unlike the well known Denavit-Hartenberg [Denavit and Hartenberg, 1955] notation) or by using regular reference frames (3 translations + 3 rotations). The robot is defined as a tree structure from the base up to each end-effector or wheel. For robots with kinematics loops, the desired chain is closed by defining an additional constraint between two branches of the tree. Similarly, declaring a constraint between any body of the structure and an input (3D device, network, etc.) will make the VirtualRobot program compute the appropriate inverse kinematic behavior of the structure to satisfy these constraints.

The use of a generic description file (see Figure 5 for an example), to define robots allows rapid simulation creation for new robot and rover structures

as well as easy modification of configurations (for example putting a new arm on a rover). For instance, a kinematic simulation of the Russian six-wheeled rover Marsokhod with four articulations (see model in Figure 1) was completed in less than a day.

Generic kinematic solver

The robot structure is represented internally by a directed graph. The nodes of the graph are the joints, bodies, input devices or external inputs and the edges of the graph express the constraints between these elements. The core solver of VirtualRobot traverses this graph and builds the "Augmented Jacobian" of the robot. The Augmented Jacobian is a matrix (usually non-square) mapping the joint velocities to the Cartesian velocities (usually of the end-effector or wheels) for each branch of the structure. The closed-loop kinematic chains are also included in the Augmented Jacobian which then maps the joint velocities to an error vector used to "close" each loop. To solve the constraints governing the robot, the Augmented Jacobian is inverted to find each joint velocity given a set of inputs. VirtualRobot computes the pseudo-inverse of the Augmented Jacobian by doing a Singular Value Decomposition (SVD). The SVD produces good solutions in most cases and provides additional information on the proximity of a singularity[2].

The kinematic solver of VirtualRobot computes in real time the kinematic behavior of any serial or parallel, under- or over-actuated structure. The rover case is handled as a multi-branch manipulator with a free-floating base. The rover is controlled by giving displacement vectors for each wheel. The displacement vector is a combination of the wheel speed and an error vector to "stick" the wheel to the terrain. A fast algorithm based on a modified elevation map has been implemented to guarantee the performance.

Customization of VirtualRobot for VIPER

The VirtualRobot system was extended, as described above, to accommodate wheeled vehicles moving on a surface. The specific extension needed to incorporate VirtualRobot into VIPER was to translate internal parameters into messages to send to Viz and telemetry to send to the plan executive. Communication from the plan executive to VirtualRobot was accomplished via its existing ability to receive inputs from the network. The total effort required was a few days.

3.3 Conditional Executive

Throughout a mission, detailed mission operations plans must be constructed, validated, and uplinked to a rover. In current practice, a mission operations plan takes the form of a rigid, time-stamped sequence of low-level commands. Unfortunately, there is uncertainty about many aspects of task execution: exactly how long operations will take, how much power will be consumed, and how much data storage will be needed. Furthermore, there is uncertainty about environmental factors that influence such things as rate of battery charging or which scientific tasks are possible. In order to allow for this uncertainty, current plans are based on worst-case estimates and contain fail-safe checks. If a task takes less time than expected, the spacecraft or rover just waits for the next time-stamped task. If a task takes longer than expected, it may be terminated before completion, potentially halting all non-essential operations until a new command plan is received. Either of these situations results in unnecessary delays and lost science opportunities.

To express plans that will handle these sources of execution uncertainty, we have designed the Contingent Rover Language (CRL), a flexible, conditional sequence language [Bresina *et al.*, 1999]. as well as the Conditional Executive (CX). CRL expresses time constraints and state constraints on the plan, but allows flexibility in the precise time that actions must be executed. Constraints include conditions that must hold before, during, and after actions are executed; failure of these conditions leads to an execution failure and potential plan adaptation.

CRL

The basic element of a CRL plan is a *node*, which may be a single action, a choice point, or a composite action that is implemented by a CRL subplan. This provides a hierarchical structure, where constraints may be placed on high-level actions as well as individual actions. The plan may contain branches to handle contingencies or opportunities that arise during execution; branches may also specify alternative courses of action for achieving the mission goals. Within any contingent branch there may be further contingent branches, so a plan is a tree of alternative courses of action. See Figure 6 for excerpts of a plan. At this time, CRL and the executive do not support concurrent activities, so there is a single path of execution through the plan structure. This is sufficient for current planetary rover operations, but as rovers grow in complexity, there may be an opportunity and need for concurrent action execution.

CX

CX is responsible for interpreting the command plan coming from "mission control," checking run-time resource requirements and availability, monitoring plan execution, and potentially selecting alternative plan branches if the situation changes [Washington *et al.*, 1999].

The input to CX consists of the primary plan and a set of alternate plans in CRL, as described above. CX executes each node while verifying that the preconditions, maintenance conditions, and end condi-

[2]VirtualRobot implements in addition a "Singular Robust Inverse" algorithm to avoid falling into singularities.

```
(block :id plan
       :node-list
       ((task :id drive-1a  :comment "drive 2.8m"
              :action baseDrive   :parameters (0.2 0.0 0 2.8))
        (task :id turn-1b   :comment "turn right 75 degs"
              :action baseDrive   :parameters (0.0 0.3 0 1.309))
        (task :id drive-1c  :comment "drive 0.5m"
              :action baseDrive   :parameters (0.2 0.0 0 0.5))
        (task :id mosaic-1  :comment  "take mosaic of Yogi"
              :action icmosaic
              :parameters ("test1/images/pan1/p3_003"
                           3 0 0 640 480 3
                           0.1 -0.8 0.7 -0.4 0.2 0.3 1))
        (branch :id branch-on-time
                :comment "chooses route based on time"
                :options
                ((option :id opt1
                         :eligible-conditions ((time 0 60))
                         :utility 1.0
                         :node
                         (block :id branch-1
                                :node-list
                                ((task :id drive-2a
                                       :comment "drive -0.5m"
                                       :action baseDrive
                                       :parameters (-0.2 0.0 0 0.5))
                                 ...
                                 )))
                 (option :id opt2
                         :eligible-conditions
                         ((time 60 :plus-infinity))
                         :utility 0.5
                         :node
                         (block :id branch-2
                                :node-list
                                (
                                 ...
                                 )))
                ))))
```

Figure 6: Excerpts of a CRL plan.

tions are respected. CX can be instructed to wait for a subset of the preconditions (for example, the start time window) rather than failing the execution of the node. CX receives state information from the low-level rover control (or simulation). In the envisioned overall rover architecture, CX will also receive higher-level state information from a state-identification module and resource information from a resource manager. It uses this information to check preconditions and maintenance conditions, as well as to check the eligibility conditions of the plans in the alternate plan library.

At each point in time, CX may have multiple options, corresponding to the eligible options of a branch point and the enabled alternate plans. CX chooses the option with the highest estimated expected utility, computed over the remainder of the plan. In the current implementation, the utility of successfully completing an atomic action is fixed and set by operators at mission control. From this atomic utility and a model of the probabilities of various events (such as a traverse taking longer than anticipated), the expected utility of an entire branching plan can be calculated.

When plan execution fails, CX reacts as specified in the node, either ignoring the action or aborting the execution and checking for applicable alternate plans. If execution is aborted and no alternate plans apply, CX aborts the entire plan set and puts the rover into a stable standby mode; all operation is suspended and the rover awaits further plans from mission control.

CX and CRL were incorporated into the rover autonomy architecture used to control the Marsokhod rover during a February 1999 field test [Bresina *et al.*, 1999] and the K9 rover during a May 2000 field test [Bresina *et al.*, 2001].

Customization of CX and CRL for VIPER

The CRL grammar is generic; only the command and condition names change from one application to another. The CX execution semantics rely on the general CRL properties and not the command and condition names. As such, the central "execution engine" is completely generic. The only specializations needed are in terms of communication with the externally controlled rover (or simulation). These are accomplished with C++ subclassing that is completely transparent to the execution engine.

In order to control the K9 rover, the CRL "command dictionary" was defined, as well as routines that CX calls to translate CRL commands to messages to the rover. To incorporate CX and CRL into VIPER, the same command dictionary was used, but the translation routines were changed to communicate with the simulation rather than the actual rover. In all, the total effort was no more than a week, much of that to separate out the parts common to the actual and simulated rover to avoid code duplication.

4 Illustrative Example

Consider a simulation of a rover moving in the Mars Pathfinder environment. VIPER presents the viewer with three main windows that show: 1) a 3D visualization of a mobile robot at the Mars Pathfinder landing site executing a plan, 2) a display showing the VirtualRobot parameters, and 3) a 2D text display of the robot executive status. See Figure 2 for representative screen images.

4.1 Scenarios

Consider the scenario where the rover is located next to the lander, as in Figure 7. The next goals to achieve may include getting a close-up mosaic of Yogi, the largest rock in the environment, followed by a traverse around the lander to near the second ramp (Ramp2). Depending on the data storage available after the mosaic ends, the rover may decide to traverse via either side of the lander; one has more rocks that are interesting scientifically, the other has fewer. In either case images will be acquired of the targets during the traverse. If the time is too short, the rover will remain near the first lander ramp (Ramp1).

These three main scenarios, shown schematically in Figure 8 and graphically in Figure 7, are the result of data, power, and time limitations on plan execution.

The first scenario illustrates the effects of a time shortage in the absence of any data storage short-

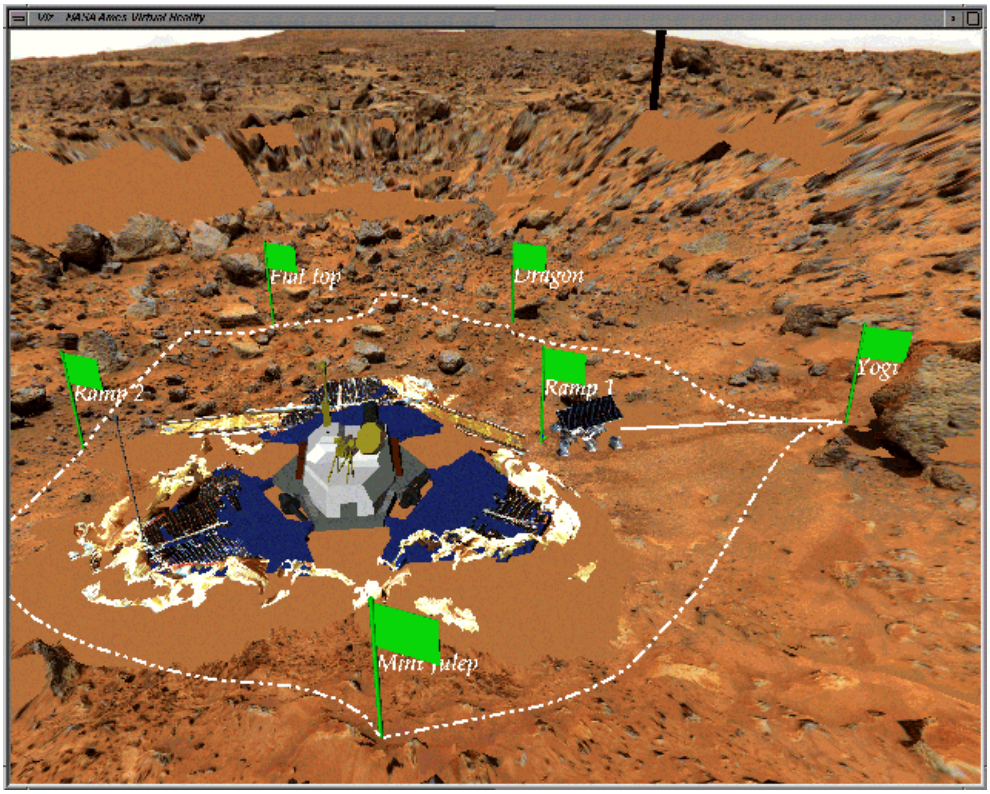

Figure 7: Side view of rover traverses.

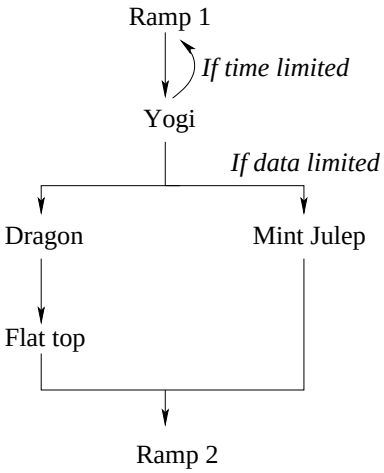

Figure 8: Possible plan branches.

age. The rover will traverse to a single location and return immediately. From the "Ramp 1" location, the rover will proceed to the "Yogi" location and return directly to "Ramp 1."

The second scenario is the situation when there are no data or time limitations. From the "Ramp 1" location, the rover will proceed to the "Yogi" location, continue on to "Dragon," "Flat top," and finally "Ramp 2."

The final scenario is the result of a data storage shortage. The rover will stop at only two locations to take images because of the resource limitations. From the "Ramp 1" location, the rover will proceed to the "Yogi" location, "Mint Julep," and finally "Ramp 2."

5 Current Directions

In parallel to the development of VIPER, a new project called the Mission Simulation Facility (MSF) has been started. The Mission Simulation Facility is being developed as a simulation environment where developers of autonomous control systems can test their system's performance against a set of integrated simulations of NASA mission scenarios.

The MSF will propose a common framework, built on top of the standardized High Level Architecture (HLA) [DMSO, 2000], allowing the different components of the simulation to interact together in a structured and extensible manner. In addition, MSF will provide a common set of messaging facilities allowing developers of autonomous software to interact with the simulation in the same manner as they would with a physical robotic system.

The MSF framework will be first applied to the VIPER application. It will unify the communication between the different components (Conditional Executive, VirtualRobot and VIZ) and at the same

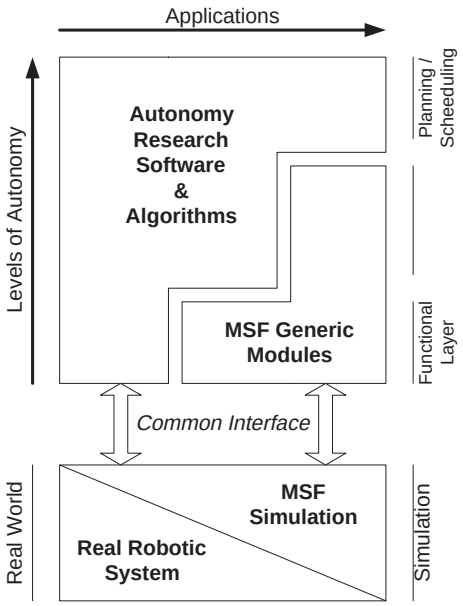

Figure 9: Scope of MSF.

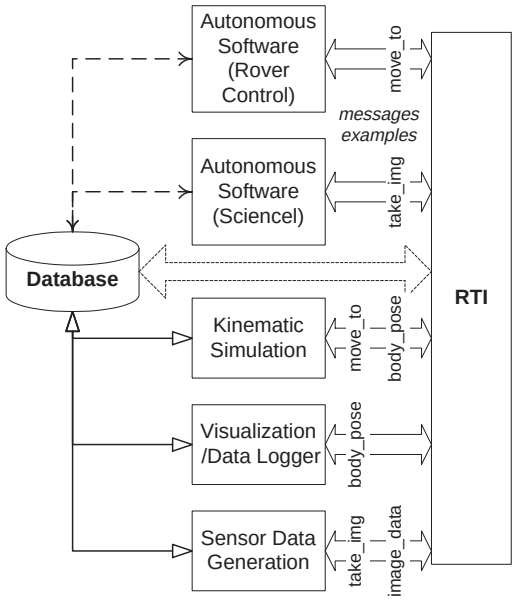

Figure 10: MSF Sample Simulation Instantiation.

time simplify their integration into a more reliable simulation.

As shown in Figure 9, MSF will provide simulation environments which could be used at different level of integration. For example a research lab could have a complete system from the top level autonomy to the low level hardware control: in this case MSF will only provide a replacement for the robotic hardware and the environment with the simulation. On the other hand, one could need to test a very high level autonomy component, without having the rest of the system: in this case, MSF will also provide generic components replacing the missing parts.

From an implementation point of view, MSF will rely on the publish/subscribe scheme used in HLA: each component of the simulation will communicate with the other components using a standard set of objects/messages defined for the purpose of MSF. The HLA Run-Time-Infrastructure (RTI) manages all the communications between the participants of the simulation. The RTI also provides facilities to address the problem of Time-Management which is a key point in a simulation like MSF because its different components are not necessarily designed to run in real time.

The Figure 10 shows a simple simulation example: it is composed of several components which are all connected to the RTI for communication. They also have access to a separate database to avoid overloading the network when accessing large objects like images. The autonomous software is decomposed into two distinct objects for clarity: each uses a different set of messages to interact with the simulation. Consider in this example that they send commands to a simulated robot. The kinematic simulator (like VirtualRobot) will then compute the behavior of the robot in response to the commands and return various sensor updates. If the autonomous software requests the acquisition of scientific data (like an image), then the sensor data generator will return either simulated data or real data extracted from a database. Visualization tools (like VIZ) could also be connected to the simulation to help the user evaluate the behavior of the autonomous system.

The development of MSF will provide a set of tools (simulation components / robotic systems library) that should speed up the testing process for the developers of autonomous systems. In addition, the proposed framework will help advance the development of standardization of communication between autonomous software and robotic hardware.

References

[Bares and Wettergreen, 1999] J. Bares and D. Wettergreen. Dante II: Technical description, results and lessons learned. *International Journal of Robotics Research*, 18(7):621–649, July 1999.

[Bernard *et al.*, 1998] D. E. Bernard, G. A. Dorais, C. Fry, E. B. Gamble Jr., B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, P. P. Nayak, B. Pell, K. Rajan, N. Rouquette, B. Smith, and B. C. Williams. Design of the remote agent experiment for spacecraft autonomy. In *Proceedings of the 1998 IEEE Aerospace Conference*, 1998.

[Bresina *et al.*, 1999] J. L. Bresina, K. Golden, D. E. Smith, and R. Washington. Increased flexibility and robustness of Mars rovers. In *Proceedings of i-SAIRAS '99, The 5th International Symposium on*

Artificial Intelligence, Robotics and Automation in Space, 1999.

[Bresina *et al.*, 2001] J. L. Bresina, M. G. Bualat, L. J. Edwards, R. M. Washington, and A. R. Wright. K9 operations in May '00 dual-rover field experiment. In *Proceedings of i-SAIRAS '01, The 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.

[Christian *et al.*, 1997] D. Christian, D. Wettergreen, M. Bualat, K. Schwehr, D. Tucker, and E. Zbinden. Field experiments with the ames marsokhod rover. In *Proceedings of the 1997 Field and Service Robotics Conference*, December 1997.

[Denavit and Hartenberg, 1955] J. Denavit and R. Hartenberg. "a kinematic notation for lower pair mechanism based on matrices. *Journal of Applied Mechanics*, 22(2):215–221, 1955.

[DMSO, 2000] DMSO, 2000. HLA official web page, URL: http://www.dmso.mil/index.php?page=64.

[Flückiger *et al.*, 1998] L. Flückiger, C. Baur, and R. Clavel. CINEGEN: a rapid prototyping tool for robot manipulators. In Gerhard Schweitzer, Roland Siegwart, and Philippe Cattin, editors, *The Fourth International Conference on Motion and Vibration Control (MOVIC'98)*, volume 1, pages 129–134, Zürich, Switzerland, 1998.

[Flückiger, 1998] L. Flückiger. *Interface pour le pilotage et l'analyse des robots basée sur un générateur de cinématiques générales.* (in french), EPFL (Swiss Federal Institute of Technology, Lausanne), Switzerland, 1998.

[Khalil and Kleinfinger, 1986] W. Khalil and J. F. Kleinfinger. A new geometric notation for open and closed-loop robots. In IEEE, editor, *Conference on Robotics and Automation*, pages 1174–1180, San Francisco, April 1986.

[Muscettola *et al.*, 1998] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1/2), August 1998.

[Nguyen *et al.*, 2001] L. Nguyen, M. Bualat, L. Edwards, L. Flueckiger, C. Neveu, K. Schwehr, M. D. Wagner, and E. Zbinden. Virtual reality interfaces for visualization and control of remote vehicles. *Autonomous Robots*, 11(1), 2001.

[Piguet *et al.*, 1995] L. Piguet, B. Hine, P. Hontalas, and E. Nygren. VEVI: A virtual reality tool for robotic planetary exploration. In *Proceedings of Virtual Reality World*, February 1995.

[Stoker *et al.*, 1998] C. Stoker, T. Blackmon, J. Hagen, B. Kanefsky, D. Rasmussen, K. Schwehr, M. Sims, and E. Zbinden. MARSMAP: An interactive virtual reality model of the Pathfinder landing site. In *Proceedings of the Lunar and Pletary Science Conference*, 1998.

[Washington *et al.*, 1999] R. Washington, K. Golden, J. Bresina, D. E. Smith, C. Anderson, and T. Smith. Autonomous rovers for mars exploration. In *Proceedings of The 1999 IEEE Aerospace Conference*, 1999.